

Laboratorio Nro. 3

Listas Enlazadas (*Linked List*) y Listas Hechas con Arreglos (*ArrayList*)



Objetivos: 1. Escribir programas que utilicen las siguientes estructuras de datos: arreglos, cadenas, listas enlazadas, colas, pilas, conjuntos y mapas. 2. Comparar y contrastar las ventajas y desventajas de implementaciones dinámicas y estáticas de estructuras de datos. 3. Escoger la estructura de datos apropiada para solucionar un problema



Consideraciones: Lean y verifiquen las consideraciones de entrega,



Leer la Guía



Trabajo en Parejas



Si tienen reclamos,
regístrenlos en
<http://bit.ly/2g4TTKf>



Ver calificaciones
en Eafit
Interactiva



En el GitHub
docente,
encontrarán la
traducción de
los Ejercicios
en Línea

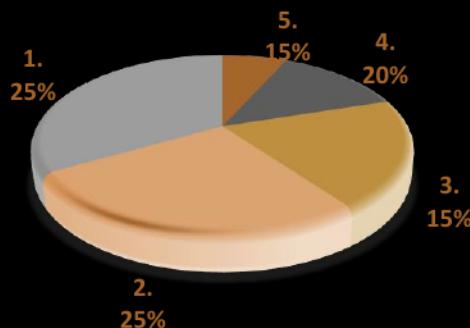


Hoy, plazo de
entrega



Subir el **informe pdf** en la carpeta **informe**, el **código del ejercicio 1** en la
carpeta **codigo** y el **código del 2** en la carpeta **ejercicioEnLinea**

Porcentajes y criterios de evaluación



- 1. Simulacro sustentación proyecto
- 2. Análisis de complejidad
- 3. Código de laboratorio
- 4. Simulacro de parcial
- 5. Ejercicio con juez en línea

1. Simulacro de Proyecto

Códigos para entregar en GitHub en la carpeta código

1 Simulacro de Proyecto Códigos para entregar en GitHub en la carpeta código



En la vida real, la documentación de software hace parte de muchos estándares de calidad como CMMI e ISO/IEC 9126



Vean Guía numeral 3.4



Código del ejercicio en línea en **GitHub**. Vean Guía en numeral 4.24



Documentación opcional. Si lo hacen, utilicen **Javadoc** o equivalente. No suban el HTML a GitHub.



No se reciben archivos en **.RAR** ni en **.ZIP**



Utilicen Java, C++ o Python



1.1 Las universidades guardan datos de las notas que obtuvo cada estudiante, en cada curso, en todos los semestres que ha cursado.

Además de cumplir con lo establecido por la ley y poder dar certificados de notas a los estudiantes, estos datos pueden utilizarse para hacer estudios de deserción, pronósticos e, incluso, evaluar la efectividad de cambios curriculares o estrategias pedagógicas.

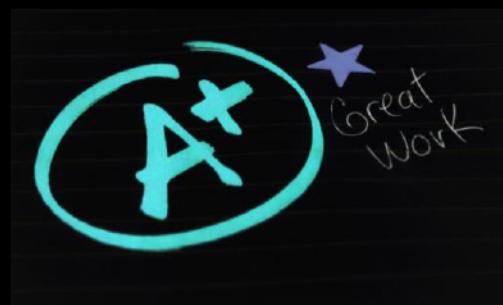


Imagen extraída de <https://bit.ly/2ThpLw9>

PhD. Mauricio Toro Bermúdez

Docente | Escuela de Ingeniería | Informática y Sistemas

Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627

Tel: (+57) (4) 261 95 00 Ext. 9473

**UNIVERSIDAD
EAFIT**

**Acreditación
Institucional**
Renovación
2018 - 2026
Resolución MEN 2158 de 2018

ESTRUCTURA DE DATOS 1
Código ST0245

► El problema a resolver consiste en **diseñar una estructura de datos** para almacenar las notas que obtuvo cada estudiante de la universidad, en cada curso, en todos los semestres que ha cursado.

El objetivo de la estructura de datos es cargar los datos CSV y realizar las siguientes consultas:

Consulta 1: Dado un curso (por ejemplo, Estructuras de Datos 1) y un semestre del año (por ejemplo 2016-2), **mostrar todos los estudiantes matriculados y la nota final** de cada uno.

Consulta 2: Dado un estudiante y un semestre del año, **dicir todos los cursos que matriculó y la nota que obtuvo** en cada uno de ellos.

La **estructura de datos debe ser eficiente en el consumo de memoria y eficiente en la complejidad asintótica** de las 2 operaciones descritas



Ejemplo: El Cuadro 1 muestra un ejemplo de la consulta 1 y el Cuadro 2 un ejemplo de la consulta 2.

Cuadro 1, ejemplo de la respuesta a la primera consulta a la estructura de datos:

Estructuras 1	
Estudiante 1	5.0
Estudiante 2	3.0
...	

Cuadro 2, ejemplo de la respuesta a la segunda consulta a la estructura de datos:

Estudiante 1	Estructuras 1	Cálculo 2	Lenguajes
	5.0	4.5	3.0

Los archivos NOTAS ST0242.csv, NOTAS ST0245.csv y NOTAS ST0242.csv contienen históricos de notas anonimizados de estudiantes de Eafit de los cursos de Fundamentos de Programación, Estructuras de Datos 1 y Estructuras de Datos 2, respectivamente.



En la vida real, balancear los pesos sobre una viga se conoce como equilibrio estático, y es de interés en Ingeniería Mecánica e Ingeniería Civil

- 1.2 [Opc]** Teniendo en cuenta lo anterior, escriban un método que reciba una lista y calcule cuál es la posición óptima de la lista para colocar un pivote



Nota: Este algoritmo se puede hacer complejidad $O(n)$ donde n es el número de elementos en la lista.



En la vida real, las listas enlazadas se usan para representar objetos en videojuegos (Ver más en <http://bit.ly/2mcGa5w>) y para modelar pistas en juegos de rol (Ver más en <http://bit.ly/2IPyXGC>)

- 1.3 [Opc]** Se tiene un almacén donde se encuentran las neveras fabricadas por una planta:

- Las primeras neveras que fueron fabricadas están de últimas dentro del almacén; las últimas neveras fabricadas recientemente, aparecen más cerca de la puerta del almacén.
- Para sacar una nevera que está atrás, primero habría que quitar la que está adelante.
- Los datos de cada nevera son el código (representado por un número) y la descripción (representada por un texto).
- El almacén dispone de una sola puerta por donde las neveras entran las neveras para ser almacenadas y por donde salen las neveras que se van a distribuir a las tiendas.
- Se tiene una lista de solicitudes de neveras, realizadas por las tiendas, donde aparece el nombre de la tienda y la cantidad solicitada de neveras. Se deben atender primero las solicitudes más antiguas y luego las más recientes.

ESTRUCTURA DE DATOS 1

Código ST0245

► Teniendo en cuenta lo anterior, elaboren un programa que reciba las neveras, en la forma en que están ordenadas en el almacén, y las solicitudes, según el orden en que llegaron, y que imprima qué neveras del almacén quedan asignadas a cada tienda. Utilice un método así:

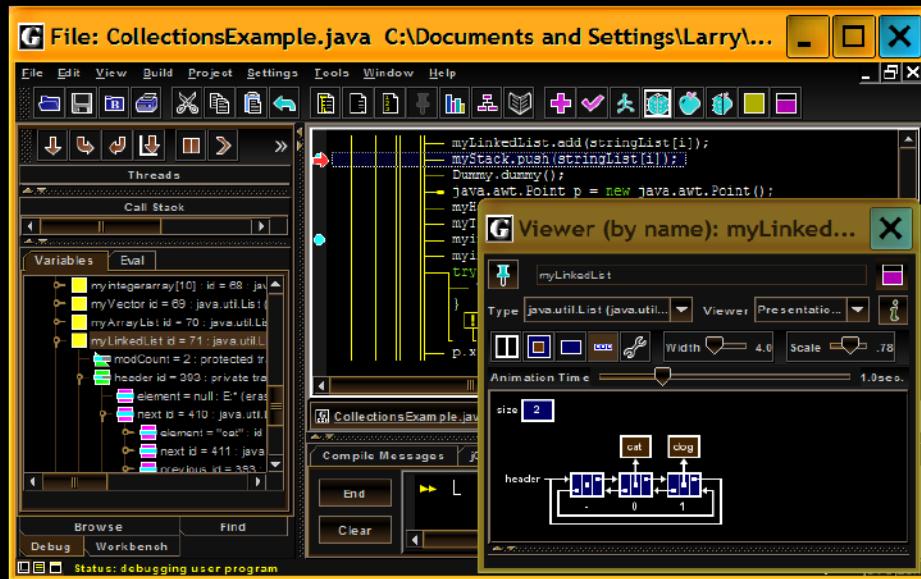
```
public static void ejercicio4( ??? neveras, ??? solicitudes)
```

! **Nota:** Este algoritmo se puede hacer complejidad $O(n.m)$ donde n es el número de solicitudes y m el máximo número de neveras en una solicitud.

1.4 [Opc] Implementen los métodos insertar un elemento en una posición determinada (conocido como *insert* o *add*), borrar un dato en una posición determinada (conocido como *remove*) y verificar si un dato está en la lista (conocido como *contains*) en la clase *LinkedListMauricio*, teniendo en cuenta que sea una lista **DOBLEMENTE** enlazada.

! **Nota 1:** Utilice el IDE Jgrasp (<http://www.jgrasp.org/>) porque tiene un depurador gráfico excelente para estructuras de datos.

! **Nota 2:** Véase a continuación gráfica de Jgrasp para efectos de ejemplificación



PhD. Mauricio Toro Bermúdez

Docente | Escuela de Ingeniería | Informática y Sistemas

Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627

Tel: (+57) (4) 261 95 00 Ext. 9473

1.5 [Opc] En el código entregado por el profesor está el método *get* y sus pruebas. Teniendo en cuenta esto, realicen 3 *tests* unitarios para cada método. La idea es probar que su implementación de los métodos *insert* y *remove* funcionan correctamente por los menos en los siguientes casos.

- Cuando vamos a *insertar/borrar* y la lista está vacía,
- Cuando vamos *insertar/borrar* el primer el elemento,
- Cuando vamos a *insertar/borrar* el último elemento.

1.6 [Opc] Teniendo en cuenta lo anterior, resuelvan lo siguiente:

En un banco hay 4 filas de clientes y solamente 2 cajeros. **Se necesita simular cómo son atendidos los clientes por los cajeros.** Si lo desean, usen su implementación de listas enlazadas; de lo contrario, use la del API de Java

▶ Escriban un método que reciba las 4 filas de personas. No se sabe la longitud máxima que puede tener una fila de clientes. Representen las filas de clientes como mejor corresponda.

- El método debe hacer una simulación y mostrar en qué cajero (1 o 2) se atiende a cada cliente de cada fila. Coloquen el método dentro de una clase *Banco*. Impriman los datos de la simulación en pantalla.
- El cajero uno se identifica con el número 1, el cajero dos con el 2. Los clientes se identifican con su nombre.
- El orden en que se atienden los clientes en cada ronda es el siguiente: primero el de la fila 1, luego el de la fila 2, luego el de la fila 3, y finalmente el de la fila 4.

Los cajeros funcionan de la siguiente forma en cada ronda: primero el cajero 1 atiende un cliente, luego el cajero 2 atiende un cliente. Se hacen rondas hasta que no queden más clientes. Si no hay clientes en una fila, se pasa a la siguiente.

2. Simulacro de Maratón de Programación sin documentación HTML, en GitHub, en la carpeta ejercicioEnLinea

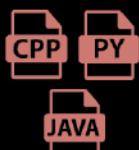
2 Simulacro de Maratón de Programación sin documentación HTML, en GitHub, en la carpeta ejercicioEnLinea



Vean Guía
numeral 3.3



No se requiere
documentación para
los ejercicios en línea



Utilicen Java,
C++ o Python



No se reciben archivos
en .PDF ni .TXT



Código del
ejercicio en línea
en **GitHub**. Vean
Guía en **numeral**
4.24

! **Nota:** Si toman la respuesta de alguna fuente, deben referenciar según el tipo de cita. Vean Guía en **numerales 4.16 y 4.17**

2.1 Resuelvan el siguiente ejercicio usando listas enlazadas:

Estás escribiendo un texto largo con un teclado roto. Bueno, no está tan roto. El único problema es que algunas veces la tecla “Inicio” o la tecla “Fin” se presionan solas (internamente).

Usted no es consciente de este problema, ya que está concentrado en el texto y ni siquiera mira el monitor. Luego de que usted termina de escribir puede ver un texto en la pantalla (si decide mirarlo).

ESTRUCTURA DE DATOS 1

Código ST0245



Habrá varios casos de prueba. Cada caso de prueba es una sola línea conteniendo por lo menos uno y como máximo 100 000 caracteres: letras, guiones bajos, y dos caracteres especiales '[' y ']'. '[' significa que la tecla "Inicio" fue presionada internamente, y ']' significa que la tecla 'Fin' fue presionada internamente. El input se finaliza con un fin-de-línea (EOF).



Por cada caso de prueba imprima la forma en que quedaría el texto teniendo en cuenta que cuando se presiona la tecla 'Inicio' lo manda al principio de la línea, y 'Fin' al final de esta. Asuma que todo el texto está en una sola línea.



```
This_is_a_[Beiju]_text
[][][]Happy_Birthday_to_Tsinghua_University
asd[fgh[jkl
asd[fgh[jkl[
[[a][d[f[[g[g[h|h[dgd[fgsfa[f
asd[gfh[[dfh]hgh]fdfhd[dfg[d]g[d]dg
```



```
BejuThis_is_a_text
Happy_Birthday_to_Tsinghua_University
jklfghasd
jklfghasd
ffgsfadgdhhggfda
dddऀgdঁhঁgঁfhasdhঁhঁfঁdঁhঁgঁdঁg
```

2.2 [Opc] Resuelvan el siguiente problema usando pilas:

Antecedentes

En muchas áreas de la ingeniería de sistemas se usan dominios simples, abstractos para tanto estudios analíticos como estudios empíricos. Por ejemplo, un estudio de IA (inteligencia artificial) en plantación y robótica (STRIPS) usó un mundo de bloques en donde un brazo robótico realizaba tareas que involucraban manipulación de bloques.

En este problema usted va a modelar un mundo de bloques simple bajo ciertas reglas y restricciones. En vez de determinar cómo alcanzar un cierto estado, usted va a “programar” un brazo robótico para que responda a un cierto conjunto de comandos.

El Problema

El problema es interpretar una serie de comandos que dan instrucciones a un brazo robótico sobre como manipular bloques que están sobre una mesa plana. Inicialmente hay n bloques en la mesa (enumerados de 0 a $n-1$) con el bloque b_i adyacente al bloque b_{i+1} para todo $0 \leq i < n-1$ tal y como se muestra en el siguiente diagrama:



Figura 1: Configuración inicial de los bloques de mesa

Los comandos válidos para el brazo robótico que manipula los bloques son:

- **move a onto b:** donde a y b son números de bloques, pone el bloque a encima del bloque b luego de devolver cualquier bloque que estén apilados sobre los bloques a y b a sus posiciones iniciales.
- **move a over b:** donde a y b son números de bloques, pone el bloque a encima de la pila que contiene al bloque b , luego de retornar cualquier bloque que está apilado sobre el bloque a a su posición inicial.
- **pile a onto b:** donde a y b son números de bloques, mueve la pila de bloques que consiste en el bloque a y todos los bloques apilados sobre este, encima de b .

PhD. Mauricio Toro Bermúdez

Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

ESTRUCTURA DE DATOS 1

Código ST0245

Todos los bloques encima del bloque *b* son movidos a su posición inicial antes de que se dé el apilamiento. Los bloques apilados sobre el bloque *a* conservan su orden original luego de ser movidos.

- **pile *a* over *b*:** donde *a* y *b* son números de bloques, pone la pila de bloques que consiste en el bloque *a* y todos los bloques que están apilados sobre este, encima de la pila que contiene al bloque *b*. Los bloques apilados sobre el bloque *a* conservan su orden original luego de ser movidos.
- **quit:** termina las manipulaciones en el mundo de bloques

Cualquier comando en donde $a = b$ o en donde *a* y *b* están en la misma pila de bloques es un comando ilegal. Todo comando ilegal debe ser ignorado y no debe afectar la configuración de los bloques.



La entrada

La entrada inicia con un entero *n* sólo en una línea representando el número de bloques en el mundo de bloques. Asuma que $0 < n < 25$.

Este número es seguido por una secuencia de comandos de bloques, un comando por línea. Su programa debe procesar todos los comandos hasta que encuentre el comando quit.

Asuma que todos los comandos tendrán la forma especificada arriba. No se le darán comandos sintácticamente incorrectos.



La salida

La salida deberá consistir en el estado final del mundo de bloques. Cada posición original de los bloques enumerada $0 \leq i < n-1$ (donde *n* es el número de bloques) deberá aparecer seguida inmediatamente de dos puntos (:).

ESTRUCTURA DE DATOS 1
Código ST0245

Si hay por lo menos un bloque en esta posición, los dos puntos deberán estar seguidos de un espacio, seguido de una lista de bloques que aparece apilada en esa posición con el número de cada bloque separado de los demás por un espacio. No ponga espacios delante de las líneas.

Deberá imprimir una línea por cada posición (es decir, habrá n líneas de salida donde n es el entero en la primera línea de la salida)



Ejemplo de entrada

```
10
move 9 onto 1
move 8 over 1
move 7 over 1
move 6 over 1
pile 8 over 6
pile 8 over 5
move 2 over 1
move 4 over 9
quit
```



Ejemplo de salida

```
0: 0
1: 1 9 2 4
2:
3: 3
4:
5: 5 8 7 6
6:
7:
8:
9:
```

2.3 [Opc] Resuelvan el siguiente problema <http://bit.ly/2j9D1VF>

2.4 [Opc] Resuelvan el siguiente problema <https://goo.gl/WVXJPx>

3. Simulacro de preguntas de sustentación de Proyecto en la carpeta informe

3 Simulacro de preguntas de sustentación de Proyecto en la carpeta informe



Vean **Guía** numeral 3.4



Exporten y entreguen informe de laboratorio en **PDF, en español o Inglés**



Si hacen el **informe en español**, usen la **plantilla en español**



No apliquen **Normas ICONTEC** para esto

Si hacen el **informe en inglés**, usen a **plantilla en inglés**



En la vida real, es necesario distinguir cuándo el comportamiento asintótico de un método es constante, lineal o cuadrático, porque la diferencia es enorme para grandes volúmenes de datos como los que se utilizan en análisis de la bolsa de valores, redes sociales y mercadeo

Sobre el Simulacro de Proyectos

- 3.1** Calculen la complejidad de cada ejercicio con cada implementación de listas. Es decir, hagan una tabla, en cada fila coloquen el número del ejercicio, en una columna la complejidad de ese ejercicio usando *ArrayList* y en la otra columna la complejidad de ese ejercicio usando *LinkedList*.

¿En qué ejercicios es mejor usar una estructura o la otra? ¿Es alguna de las dos eficiente para este problema o se necesitará otra estructura de datos aún más eficiente?

ESTRUCTURA DE DATOS 1
Código ST0245

	<i>ArrayList</i>	<i>LinkedList</i>
Ejercicio 1.1		
Ejercicio 1.2 (opt)		
Ejercicio 1.3 (opt)		
Ejercicio 1.4 (opt)		

Sobre el simulacro de maratón de programación

3.2  Expliquen con sus propias palabras cómo funciona la implementación del ejercicio 2.1 y [opcionalmente] el 2.2

3.3  Calculen la complejidad del ejercicio realizado en el numeral 2.1 y [opcionalmente] 2.2, y agregarla al informe PDF

3.4  Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del numeral 3.3

 **Ejemplos de su respuesta:**

“n es el número de elementos del arreglo”,
“V es el número de vértices del grafo”,
“n es el número de filas de la matriz y m el número de columnas”.

4. Simulacro de parcial en informe PDF

4 Simulacro de Parcial en el informe PDF



Para este simulacro,
agreguen **sus respuestas**
en el informe PDF.



*El día del Parcial no
tendrán computador,
JAVA o acceso a internet.*

- 4.1 [Opc]** ¿Cuál operación tiene una mayor complejidad asintótica, para el peor de los casos, en una lista simplemente enlazada?

► Elija la respuesta que considere acertada:

- a) Buscar un dato cualquiera en la lista
- b) Insertar un elemento cualquiera en la lista
- c) Las dos tienen la misma complejidad asintótica

- 4.2 [Opc]** Pepito quiere conocer el nombre de todos los productos de una tienda. Pepito diseñó un programa que imprime los elementos de la una lista enlazada. La variable n representa el tamaño de la lista. En el peor de los casos, ¿cuál es la complejidad asintótica para el algoritmo?

! **Importante:** Recuerde que el ciclo `for each` implementa iteradores que permiten obtener el siguiente (o el anterior) de una lista enlazada en tiempo constante, es decir, en $O(1)$.

```
01 public void listas(LinkedList<String> lista) {
02     for(String nombre: lista)
03         print(nombre); }
```

ESTRUCTURA DE DATOS 1
Código ST0245

- a) $O(n^2)$
- b) $O(1)$
- c) $O(n)$
- d) $O(\log n)$

4.3 En el juego de *hot potato* (conocido en Colombia como *Tingo, Tingo, Tango*), los niños hacen un círculo y pasan al vecino de la derecha, un elemento, tan rápido como puedan. En un cierto punto del juego, se detiene el paso del elemento. El niño que queda con el elemento, sale del círculo. El juego continúa hasta que sólo quede un niño.

Este problema se puede simular usando una lista. El algoritmo tiene dos entradas:

Una lista *q* con los nombres de los niños y una constante entera *num*. El algoritmo retorna el nombre de la última persona que queda en el juego, después de pasar la pelota, en cada ronda, *num* veces.

Como un ejemplo, para el círculo de niños [*Bill, David, Susan, Jane, Kent, Brad*], donde *último* es el primer niño, *Brad* el primer niño, y *num* es igual a 7, la respuesta es *Susan*.

En Java, el método *add* agrega un elemento al comienzo de una lista, y el método *remove* retira un elemento del final de una lista y retorna el elemento.

```

01 String hotPotato(LinkedList<String> q, int num)
02     while (_____)
03         for (int i = 1; i _____ num; i++)
04             q.add(_____);
05         q.remove();
06     return _____;

```



A continuación, completen los espacios restantes del código anterior:

a) Completen el espacio de la línea 02

b) Completen el espacio de la línea 03

c) Completen el espacio de la línea 04

d) Completen el espacio de la línea 06

4.4 [Opc]

El siguiente es un algoritmo invierte una lista usando como estructura auxiliar una pila:

```

01 public static LinkedList <String> invertir
        (LinkedList <String> lista) {
02     Stack <String> auxiliar = new Stack <String>();
03     while(..... > 0){
04         auxiliar.push(lista.removeFirst());
05     }
06     while(auxiliar.size() > 0) {
07         .....
08     }
09     return lista;
10 }
```



De acuerdo a lo anterior, responda las siguientes preguntas:

a)

¿Qué condición colocaría en el ciclo while de la línea 3?

b)

Completen la línea 7 de forma que el algoritmo tenga sentido

4.5

En un banco, se desea dar prioridad a las personas de edad avanzada. El ingeniero ha propuesto un algoritmo para hacer que la persona de mayor edad en la fila quede en primer lugar.

Para implementar su algoritmo, el ingeniero utiliza colas. Las colas en Java se representan mediante la interfaz Queue. Una implementación de Queue es la clase LinkedList.

El método offer inserta en una cola y el método poll retira uno elemento de una cola.

ESTRUCTURA DE DATOS 1
Código ST0245

```

01 public Queue<Integer> organizar(
02     Queue<Integer> personas) {
03     int mayorEdad = 0;
04     int edad;
05     Queue<Integer> auxiliar1 =
06         new LinkedList<Integer>();
07     Queue<Integer> auxiliar2 =
08         new LinkedList<Integer>();
09     while(personas.size() > 0){
10         edad = personas.poll();
11         if(edad > mayorEdad) mayorEdad = edad;
12         auxiliar1.offer(edad);
13         auxiliar2.offer(edad);
14     }
15     while(.....){
16         edad = auxiliar1.poll();
17         if(edad == mayorEdad) personas.offer(edad);
18     }
19     while(.....){
20         edad = auxiliar2.poll();
21         if(edad != mayorEdad) ....;
22     }
23     return personas;
24 }
```

 **Respondan las siguientes preguntas**

4.5.1 ¿Qué condiciones colocaría en los 2 ciclos while de líneas 12 y 16, respectivamente?

4.5.2 Completén la línea 18 con el objeto y el llamado a un método, de forma que el algoritmo tenga sentido

- 4.6** El ***add(n)*** añade el elemento *n* al final de la lista. El ***get(i)*** retorna el elemento en la posición *i*. El ***size()*** retorna el tamaño de la lista.

¿Cuál es la complejidad asintótica, en el peor de los casos, de la siguiente función?

```
public void imprimir(int n) {
    if (n == 1) println(1);
    else { println(n);
        imprimir(n-1);    }
}

void funcion1(LinkedList<Integer> lista){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; ++j) {
            lista.add(i * j);
        }
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++) {
            print(lista.get(j));
        }
    }
}
```

► Elija la respuesta que considere acertada:

- a) O(n^3)
- b) O(n^2)
- c) O($\log n$)
- d) O(n)

- 4.8** ¿Cuál es la complejidad asintótica, en el peor de los casos, de insertar un elemento en la posición $i, 0 \leq i < |lista|$ de una lista enlazada?

! **Nota:** $|e|$ se usa para denotar el número de elementos de la lista *e*.

► Elija la respuesta que considere acertada:

- a) $O(n^2)$.
- b) $O(\log n)$
- c) $O(n)$
- d) $O(1)$

4.9 El método add(*i*) agrega el elemento *i* al inicio de la cola. El método poll() retira el elemento al final de la cola y retorna su valor. Considere el siguiente método:

```
void calcular(int k){
    Queue<Integer> q = new Queue();
    for(int i = 0; i < k; ++i){
        if(k % 3 == 0 && i % 3 == 0){
            q.add(k - i);
        }
    }
    int j = 0;
    while(q.size() > 0){
        if(j == 3){
            System.out.println(q.poll());
            break;
        }
        q.poll();
        j++;
    }
}
```

► Elija la respuesta acertada para cada una de las preguntas:

4.8.1 ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo anterior?

- a) $O(k)$
- b) $O(n^2)$
- c) $O(n \log k)$
- d) $O(1)$

ESTRUCTURA DE DATOS 1
Código ST0245

4.8.2 ¿Qué imprime el algoritmo anterior cuando $k=21$?

- a) 6
- b) 9
- c) 12
- d) 3

4.8.3 ¿Cuál es la complejidad asintótica, en el peor de los casos, de adicionar un dato a una cola de n elementos?

- (a) $O(\log n)$
- (b) $O(n)$
- (c) $O(1)$
- (d) $O(n^2)$

4.10 El método *push(i)* ingresa el elemento i al tope de la pila. El método *pop()* retira el elemento en el tope de la pila y retorna su valor. Considere el siguiente método:

```
void metodo(int n, int x) {
    Stack<Integer> s = new Stack();
    for(int i = 0; i < n; i++) {
        if(i % 3 == 0) {
            s.push(i);
        }
    }
    int k = 0;
    while(s.size() > 0) {
        if( k == x ) {
            System.out.println(s.pop());
            break;
        }
        k = k + 2;
        s.pop();
    }
}
```

► Elija la respuesta acertada para cada una de las preguntas:

4.9.1 ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo anterior?

- a) $O(\log n)$
- b) $O(1)$
- c) $O(n^2)$
- d) $O(n)$

4.9.2 ¿Qué valor imprime el algoritmo anterior cuando $x=8$ y $n=20$?

- a) 6
- b) 8
- c) 12
- d) 3

4.9.3 ¿Cuál es la complejidad asintótica, en el peor de los casos, de encontrar un elemento en una pila con n elementos, en otras palabras, decir si un elemento está o no está en la pila?

- a) $O(1)$
- b) $O(n)$
- c) $O(\log n)$
- d) $O(n^2)$

4.11 El método ***add(n)*** añade el elemento n en la última posición de la lista. El método ***contains(n)*** retorna verdadero si n está en la lista, sino retorna falso. El método ***size()*** retorna el tamaño de la lista. El método ***get(i)*** devuelve el elemento en la posición i de la lista.

► Elijan la respuesta acertada para cada una de las preguntas:

4.10.1 ¿Cuál es la complejidad asintótica, en el peor de los casos, de la siguiente función?

ESTRUCTURA DE DATOS 1
Código ST0245

```

void fun1(ArrayList<Integer> list) {
    int a = -1, n=list.size();
    for(int i=n; i >= 0;i--){
        a = Math.max(a, list.get(i));
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < a; j++) {
            list.add(i*j);
        }
    }
}

```

- a) $O(n^2)$
- b) $O(\max(list) \times n)$
- c) $O(\max(list) \times n^2)$
- d) $O(n)$

4.10.2 ¿Cuál es la complejidad asintótica, en el peor de los casos, de insertar un elemento al principio de una lista hecha con arreglos (*ArrayList*)?

- a) $O(n^2)$
- b) $O(n)$
- c) $O(\log n)$
- d) $O(1)$

4.12 Polka desea implementar una cola de una manera muy especial: Él ya tiene implementada una pila, pero no quiere implementar una cola desde el principio, entonces él ha decidido usar la pila que ya tiene implementada para implementar la cola.

Él quedará conforme si implementa las dos funciones principales de la cola: *offer(e)* (añade el elemento e al inicio de la cola), *poll()* (extrae y elimina el primer elemento que entró a la cola).

Él ya implementó *offer(e)*, pero no ha podido con *poll()*, entonces él quiere que tú lo implementes. Las variables *s1,s2* son dos pilas que se inicializan en el constructor de la clase *Cola*.

ESTRUCTURA DE DATOS 1
Código ST0245

! **Nota 1:** En una pila, la función *pop()* retorna y extrae el elemento en el tope de la pila, y la función *push(x)* añade el elemento *x* al tope de la pila.

! **Nota 2:** En la vida real no se implementa una cola con pilas porque no es eficiente

```
class Cola{
    Stack<Integer> s1, s2;
    Cola() {
        s1 = new LinkedList<>();
        s2 = new LinkedList<>();
    }
    void offer(int e) {
        s1.push(e);
    }
    int poll(){
        //Completa por favor
        if(s2.isEmpty()){
            while(_____) //línea 13
                s2.push(______); //línea 14
        }
        return _____; //línea 17
    }
}
```

► A continuación, completen los espacios restantes del código anterior:

4.11.1 Completen el espacio de la línea 13

4.11.2. Completen la línea 14

4.11.3. Completen la línea 17

ESTRUCTURA DE DATOS 1

Código ST0245

4.13

El método `push(i)` ingresa el elemento `i` al tope de la pila. El método `pop()` retira el elemento en el tope de la pila y retorna su valor. Considere el siguiente método:

```

1 void método(Stack<Integer> s) {
2     for(int i = 10; i >= 0; i--) {
3         if(i % 2 == 0) {
4             s.push(i);
5         }
6     }
7     System.out.print(s.pop());
8     while(s.size() > 0) {
9         System.out.print(" " + s.pop());
10    }
11 }
```



Elijan la respuesta acertada para cada una de las preguntas:

4.12.1 ¿Cuál es la salida del método anterior?

- i) 10, 8, 6, 5, 4, 2, 0
- ii) 2, 4, 6, 8, 10
- iii) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- iv) 0, 2, 4, 6, 8, 10

4.12.2 ¿Cuál es la complejidad asintótica, en el peor de los casos, de añadir un nuevo elemento al tope una pila que contiene n elementos?

- i) $O(1)$
- ii) $O(n)$
- iii) $O(n * \log(n))$
- iv) $O(n^2)$

4.13

Sean L_1 y L_2 listas simplemente enlazadas, y sea x un número entero aleatorio tomado del rango $[1, n]$. Vamos a realizar las siguientes dos operaciones en las dos listas:

1. Si x NO se encuentra en la lista L_1 , entonces lo insertamos al inicio de L_1 y, además, si x se encuentra en L_2 , tomamos x números aleatorios del rango $[1, n]$ y los insertamos al inicio de L_2 .
2. Si el número x se encuentra x veces en L_2 , lo insertamos al inicio de L_1 .

Puedes asumir que encontrar un número aleatorio en el rango $[1, n]$ se hace en $O(1)$. También, puedes asumir que buscar un número en una lista simplemente enlazada se hace en $O(n)$.



Elijan la respuesta acertada para cada una de las preguntas:

4.13.1 La primera operación se ejecuta n veces. ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo que ejecuta la primera operación n veces?

- i) $O(n)$
- ii) $O(2^n)$
- iii) $O(n^2)$
- iv) $O(n^3)$

4.13.2 La segunda operación se ejecuta n veces. ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo que ejecuta la segunda operación n veces?

- i) $O(n)$
- ii) $O(2^n)$
- iii) $O(n^2)$
- iv) $O(n^3)$

4.14

El método `add(i)` agrega el elemento `i` al inicio de la cola. El método `poll()` retira el elemento al final de la cola y retorna su valor. Considere el siguiente método:

PhD. Mauricio Toro Bermúdez

Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

ESTRUCTURA DE DATOS 1
Código ST0245

```
1 void método(Queue<Integer> q) {  
2     for(int i = 2; i * i <= 25; i++) {  
3         q.add(i);  
4     }  
5     System.out.print(q.poll());  
6     while(q.size() > 0){  
7         System.out.print(" " + q.poll());  
8     }  
9 }
```



¿Cuál es la salida del método anterior?

- i) Los enteros positivos menores que 26 en orden ascendente.
- ii) Los enteros positivos menores que 26 en orden descendente.
- iii) 2, 3, 4, 5
- iv) 5, 4, 3, 2

5. [Opcional] Lecturas Recomendadas

5 [Opc] Lecturas recomendadas



Vean Guía en numeral 3.5 y 4.20



Exportar y entregar informe de laboratorio en **PDF, en español o Inglés**



Si hacen el **informe en español**, usen la **plantilla en español**



No apliquen **Icontec** para esto

Normas

Si hacen el **informe en inglés**, usen a **plantilla en inglés**



"El ejercicio de una profesión requiere la adquisición de competencias que se sustentan en procesos comunicativos. Así cuando se entrevista a un ingeniero recién egresado para un empleo, una buena parte de sus posibilidades radica en su capacidad de comunicación; pero se ha observado que esta es una de sus principales debilidades..." *Tomado de* <http://bit.ly/2gJKzJD>



Vean Guía en numerales 3.5 y 4.20



Lean a "*Narasimha Karumanchi, Data Structures and Algorithms made easy in Java, (2nd edition), 2011. Chapter 3: Linked Lists*" y sumen puntos adicionales, así:



Hagan un mapa conceptual con los principales elementos teóricos.

ESTRUCTURA DE DATOS 1
Código ST0245



Nota: Si desean una lectura adicional en inglés, consideren la siguiente: “*J Robert Lafore. Data Structures and Algorithms in Java. Chapter 5: Linked Lists*” que pueden encontrarla en biblioteca

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

UNIVERSIDAD
EAFIT® 
Resolución MEN 2158 de 2018

6. [Opcional] Trabajo en Equipo y Progreso Gradual

6 [Opc] Trabajo en Equipo y Progreso Gradual



Vean Guía en numeral 3.5 y 4.20



Si hacen el **informe en español**, usen la **plantilla en español**



Exportar y entregar informe de laboratorio en **PDF, en español o Inglés**



No apliquen **Normas Icontec** para esto

Si hacen **el informe en inglés**, usen a **plantilla en inglés**



El trabajo en equipo es una exigencia del mercado. "Mientras algunos medios retratan la programación como un trabajo solitario, pero requiere mucha comunicación y trabajo grupal. Si trabajas para una compañía, serás parte de un equipo de desarrollo y esperarán que te comuniques bien con otras personas" *Tomado de <http://bit.ly/1B6hUDp>*



Vean Guía en numerales 3.6, 4.21, 4.22, 4.23

6.1



Entreguen copia de todas las actas de reunión usando el *tablero Kanban*, con fecha, hora e integrantes que participaron

6.2



Entreguen el reporte de *git* con los cambios en el código y quién hizo cada cambio, con fecha, hora e integrantes que participaron

6.3



Entreguen el reporte de cambios del informe de laboratorio que se genera *Google docs* o herramientas similares

!

Nota: Estas respuestas también deben incluirse en el informe PDF

7. [Opcional]

**Laboratorio en Inglés con
plantilla en Inglés**

7 [Opc] Laboratorio en inglés



Vean Guía en numeral 3.5 y 4.20



Si hacen el **informe en español**, usen la **plantilla en español**

Si hacen **el informe en inglés**, usen a **plantilla en inglés**



Exportar y entregar informe de laboratorio en **PDF, en español o Inglés**



No apliquen **Normas Icontec** para esto



El inglés es un idioma importante en la Ingeniería de Sistemas porque la mayoría de los avances en tecnología se publican en este idioma y la traducción, usualmente se demora un tiempo.

Adicionalmente, dominar el inglés permite conseguir trabajos en el exterior que son muy bien remunerados. *Tomado de goo.gl/4s3LmZ*



Entreguen el código y el informe en inglés.

Resumen de ejercicios a resolver

1.1 Diseñar una estructura de datos para almacenar las notas que obtuvo cada estudiante de la universidad, en cada curso, en todos los semestres que ha cursado.

1.2 [Ejercicio Opcional] Escriban un método que reciba una lista y calcule cuál es la posición óptima de la lista para colocar un pivote

1.3 [Ejercicio Opcional] Elaboren un programa que reciba las neveras, en la forma en que están ordenadas en el almacén, y las solicitudes, según el orden en que llegaron, y que imprima qué neveras del almacén quedan asignadas a cada tienda.

1.4 [Ejercicio Opcional] Implementen los métodos insertar un elemento en una posición determinada (conocido como *insert* o *add*), borrar un dato en una posición determinada (conocido como *remove*) y verificar si un dato está en la lista (conocido como *contains*) en la clase *LinkedListMauricio*

1.5 [Ejercicio opcional] En el código entregado por el profesor está el método *get* y sus pruebas. Teniendo en cuenta esto, realicen 3 *tests* unitarios para cada método. La idea es probar que su implementación de los métodos *insert* y *remove* funcionan correctamente por lo menos en los siguientes casos

1.6 [Ejercicio opcional] En un banco hay 4 filas de clientes y solamente 2 cajeros. Se necesita simular cómo son atendidos los clientes por los cajeros. Si lo desean, usen su implementación de listas enlazadas; de lo contrario, use la del API de Java

2.1 Resuelvan el problema del teclado roto usando listas enlazadas

2.2 [Ejercicio Opcional] Resuelvan el problema de una serie de comandos que dan instrucciones a un brazo robótico sobre como manipular bloques que están sobre una mesa plana, usando pilas

2.3. [Ejercicio Opcional] Resuelvan el problema <http://bit.ly/2j9D1VF>

2.4. [Ejercicio Opcional] Resuelvan el problema <https://goo.gl/WVXJPx>

3.1 Calculen la complejidad de cada ejercicio con cada implementación de listas. Es decir, hagan una tabla, en cada fila coloquen el número del ejercicio, en una columna la complejidad de ese ejercicio usando *ArrayList* y en la otra columna la complejidad de ese ejercicio usando *LinkedList*.

ESTRUCTURA DE DATOS 1
Código ST0245

3.2 Expliquen con sus propias palabras cómo funciona la implementación del ejercicio 2.1 y [opcionalmente] el 2.2

3.3 Calculen la complejidad del ejercicio realizado en el numeral 2.1 y [opcionalmente] 2.2, y agregarla al informe PDF

3.4 Expliquen con sus palabras las variables (qué es 'n', qué es 'm', etc.) del cálculo de complejidad del numeral 3.3

4. Simulacro de Parcial

5. [Ejercicio Opcional] Lectura recomendada

6. [Ejercicio Opcional] Trabajo en Equipo y Progreso Gradual

7. [Ejercicio Opcional] Utilicen la plantilla dispuesta en este idioma para el laboratorio.

Ayudas para resolver los Ejercicios

Ayudas para el Ejercicio 1.....	<u>Pág. 42</u>
Ayudas para el Ejercicio 1.3.....	<u>Pág. 42</u>
Ayudas para el Ejercicio 1.4.....	<u>Pág. 43</u>
Ayudas para el Ejercicio 1.5.....	<u>Pág. 44</u>
Ayudas para el Ejercicio 1.6.....	<u>Pág. 47</u>
Ayudas para el Ejercicio 1.7.....	<u>Pág. 47</u>
Ayudas para el Ejercicio 2.1.....	<u>Pág. 48</u>
Ayudas para el Ejercicio 2.2.....	<u>Pág. 48</u>
Ayudas para el Ejercicio 2.3.....	<u>Pág. 49</u>
Ayudas para el Ejercicio 2.4.....	<u>Pág. 49</u>
Ayudas para el Ejercicio 3.1.....	<u>Pág. 49</u>
Ayudas para el Ejercicio 4.....	<u>Pág. 50</u>
Ayudas para el Ejercicio 5.....	<u>Pág. 51</u>
Ayudas para el Ejercicio 6.1.....	<u>Pág. 51</u>
Ayudas para el Ejercicio 6.2.....	<u>Pág. 51</u>
Ayudas para el Ejercicio 6.3.....	<u>Pág. 51</u>



Ayudas para el Ejercicio 1.1

Para el ejercicio 1.1, tengan en cuenta que:

- Si lo hicieran con un arreglo de arreglos, tendrían una matriz de 14000 estudiantes x 3000 cursos y habría muchos espacios vacíos porque 1 curso sólo tiene máximo 30 estudiantes y un estudiante tiene máximo 7 materias.
- Es más fácil abrir los archivos en formato .csv que los archivos de Excel. Se utiliza el comando *Split* para dividir las cadenas.
- Sólo se necesita guardar la nota definitiva, hacer caso omiso a las notas parciales



¿Qué tal si hacen una lista de listas, es decir *LinkedList<LinkedList<Integer>>*



Ayudas para el Ejercicio 1.2

Para el ejercicio 1.2, tengan en cuenta que:

- Si deciden usar otro lenguaje de programación, asegúrense de usar una implementación de listas enlazadas y una de listas con arreglos.
- En la mayoría de los lenguajes (*Php, Python, Ruby*), las listas, por defecto, están hechas con arreglos.
- Para cada función implementar un método que trabaje con *ArrayList* y uno que trabaje con *LinkedList* o, si es posible, un método que sirva para los dos tipos de lista (sí es posible). Utilicen las listas disponibles en el API de Java.



Pista: Utilicen la interfaz *List* de Java <http://bit.ly/2hxCaNu>



Ayudas para el Ejercicio 1.3



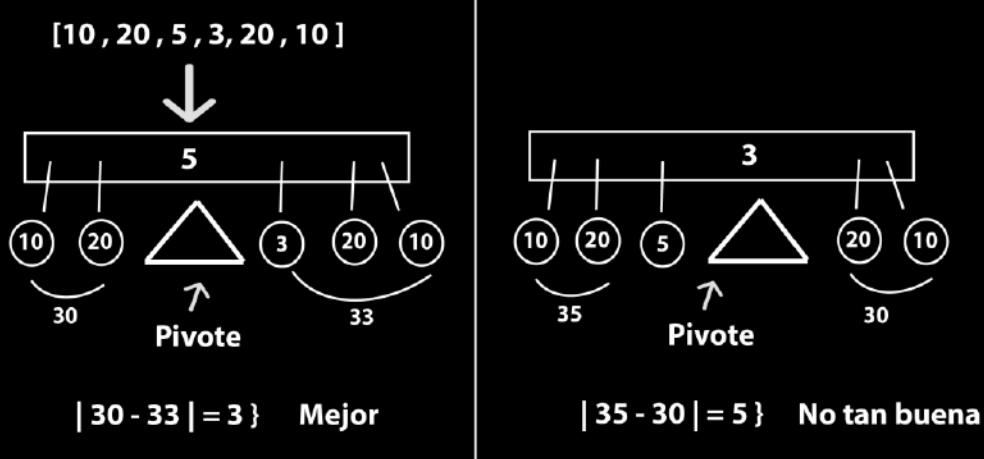
Como un ejemplo, para la lista [10, 20, 5, 3, 20, 10] la posición 2 (en donde está la materia de peso 5 kg) es el mejor pivote, porque al lado izquierdo queda un peso de 33 kg y al lado derecho 33 kg; en cambio, en otras posiciones, la suma de pesos a la izquierda y a la derecha queda más desigual



Como otro ejemplo, para la lista [10, 2, 4, 8] el pivote debería ir en la posición 1, en donde está el peso de 2 kg. Como un final ejemplo, en la lista [10, 2, 5, 2, 11], el pivote debería ir en la posición 2 en donde está el peso de 5 kg.



Pista 3:



Ayudas para el Ejercicio 1.4



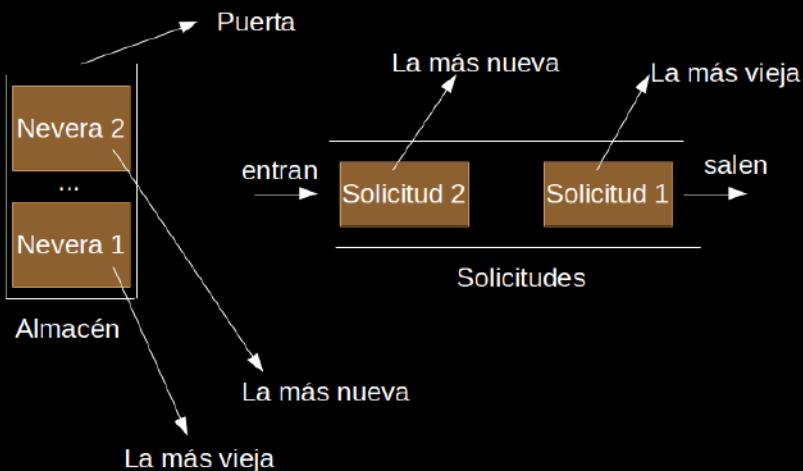
Pista 1: Utilicen listas enlazadas



Pista 2: No supongan, erróneamente, que hay suficientes neveras para cumplir las solicitudes

ESTRUCTURA DE DATOS 1

Código ST0245



Como un ejemplo, para las siguientes neveras y solicitudes,

```
almacen = [(1,"haceb"), (2,"lg"), (3,"ibm"), (4,"haceb"), (5,"lg"),
(6,"ibm"), (7,"haceb"), (8,"lg"), (9,"ibm"), (8,"lg"), (9,"ibm")]

solicitudes = [("eafit", 10), ("la14", 2), ("olimpica", 4), ("éxito",
1)]
```

Donde “éxito” fue la primera solicitud y “eafit” la última, e “ibm” con código 9 fue la última nevera ingresada a la bodega, la respuesta debe ser:

```
[('éxito', [(9, 'ibm')]),
 ('olimpica', [(8, 'lg'), (9, 'ibm'), (8, 'lg'), (7, 'haceb')]),
 ('la14', [(6, 'ibm'), (5, 'lg')]),
 ('eafit', [(4, 'haceb'), (3, 'ibm'), (2, 'lg'), (1, 'haceb')])]
```



Pista 4: Si deciden hacer la documentación, consulten la *Guía en numeral 4.1*



Ayudas para el Ejercicio 1.5



Pista 1: Diferencien *LinkedListMauricio* de *LinkedList* del API de Java. Un error común es creer que todo se soluciona llamando los métodos existentes en *LinkedList* y, no es así, la idea es implementar una lista enlazada nosotros mismos.

A continuación, un ejemplo del error:

```
// Retorna el tamaño actual de la lista
public int size()
{
    return size();
}

// Retorna el elemento en la posición index
public int get(int index)
{
    return get(index);
}

// Inserta un dato en la posición index
public void insert(int data, int index)
{
    if (index <= size())
    {
        insert(data, index);
    }
}

// Borra el dato en la posición index
public void remove(int index)
{
    remove(index);
}

// Verifica si está un dato en la lista
public boolean contains(int data)
{
    return contains(data);
}
```



Pista 2: Otro error común es pensar que todo se soluciona usando *getNode*. Sí, *getNode* es de gran utilidad, pero ¿qué pasa si *index = 0* o si la lista está vacía? A continuación, un ejemplo de cómo NO usar *getNode*.

```
// Borra el dato en la posicion index
public void remove(int index)
{
    getNode(index-1).next=getNode(index+1);
}
```



Pista 3: Otro problema común es destruir la lista sin culpa cuando uno desea consultarla. Como un ejemplo, estos métodos de `size` calculan el tamaño, pero dañan la lista, dejando la referencia `first` en `null`

```
// Malo porque daña la lista
public int size1() {
    int i = 0;
    while (first != null)
    {
        first = first.next;
        i++;
    }
    return i;
}
```

```
// Malo porque daña la lista
public int size2() {
    if (first == null)
        return 0;
    else
        first = first.next;
        return 1 + size();
}
```



Errores comunes

ESTRUCTURA DE DATOS 1
Código ST0245



Error común



Ayudas para el Ejercicio 1.6



Pista: Si deciden hacer la documentación, consulten la *Guía en numeral 4.14 y 4.15*



Ayudas para el Ejercicio 1.7



Pistas:

1. Realicen primero un dibujo de cada fila del banco y cada cajero
2. Identifiquen si la fila es una lista, cola, pila o arreglo
3. Identifiquen si el cajero es un número, una lista, una pila, una cola o un arreglo
4. Identifiquen si va a guardar las 4 filas en una lista, pila, cola o arreglo.

```
major class Laboratory4{
    public static void simular(??? filas) {
        ...
    }
}
```



Pista 5: Si deciden hacer la documentación, consulten la *Guía en numeral 4.1*



Ayudas para el Ejercicio 2.1



Pista 1: Si este ejercicio se hace en un juez en línea con una complejidad de $O(n^2)$, saldrá *time out*

Como un ejemplo, de esta forma el recorrido de la lista se convierte en $O(n^2)$ porque `get()` es $O(n)$ y está en un ciclo que se hace n veces.

ESTRUCTURA DE DATOS 1
Código ST0245

```
public static int multi(LinkedList<Integer> lista) {
    int acum = 1;
    for (int i = 0; i < lista.size(); ++i) {
        acum *= lista.get(i);
    }
    return acum;
}
```



Pista 2: Vean *Guía en numeral 4.13*



Ayudas para el Ejercicio 2.2



Pista 1: Utilicen pilas



Pista 2: Vean *Guía en numeral 4.13*



Pista 3: Lo mejor es utilizar la técnica de diseño de modularización. Entonces, escribir un método para resolver cada uno de los siguientes problemas:

- move a onto b
- move a over b
- pile a onto b
- pile a over b

Posteriormente, se crea un mundo de bloques usando una pila para cada columna. Finalmente, se lee la entrada y se ejecuta la acción que corresponda de las 4 definidas anteriormente.



Ayudas para el Ejercicio 2.3



Pista: No construya una solución $O(n^2)$ para este problema



Ayudas para el Ejercicio 2.4



Pista: Utilicen pilas



Ayudas para el Ejercicio 3.1



Pista 1: Si quieren que sean eficientes los ejercicios, usen iteradores, así:
<http://bit.ly/2cwWdbe>

Como un ejemplo, de esta forma el recorrido de la lista se convierte en $O(n^2)$ porque `get()` es $O(n)$ y está en un ciclo que se hace n veces.

```
public static int multi(LinkedList<Integer> lista) {
    int acum = 1;
    for (int i = 0; i < lista.size(); ++i) {
        acum *= lista.get(i);
    }
    return acum;
}
```



Pista 2: Recuerden que la complejidad de un algoritmo es $O(n^2)$ cuando hay 2 ciclos anidados que dependen de la misma entrada, pero si hay dos entradas, por

ESTRUCTURA DE DATOS 1

Código ST0245

ejemplo dos arreglos diferentes, y un ciclo recorre un arreglo y el otro ciclo recorre el otro arreglo, entonces la complejidad es O(n.m)



Pista 3: La complejidad de este algoritmo es O(n) porque el `for each` es capaz de acceder al siguiente elemento de una lista enlazada en tiempo constante

```
for(String nombre : lista)
    print(nombre)
```



Pista 4: Vean Guía en numeral 4.11 y 4.19



Ayudas para el Ejercicio 4



Pista 1: Vean Guía en numeral 4.18



Ayudas para el Ejercicio 5



Pista 1: Para que hagan el mapa conceptual se recomiendan herramientas como las que encuentran en <https://cacoo.com/> o <https://www.mindmup.com/#m:new-a-1437527273469>



Ayudas para el Ejercicio 6.1



Pista 1: Vean Guía en numeral 4.21



Ayudas para el Ejercicio 6.2



Pista 1: Vean Guía en numeral 4.23



Ayudas para el Ejercicio 6.3



Pista 1: Vean Guía en numeral 4.22

¿Alguna inquietud?

CONTACTO

Docente Mauricio Toro Bermúdez
Teléfono: (+57) (4) 261 95 00 Ext. 9473
Correo: mtorobe@eafit.edu.co
Oficina: 19- 627

Agenden una cita dando clic en la pestaña
-Semana- de <http://bit.ly/2gzVg10>